

PATENT ABSTRACTS OF JAPAN

(11)Publication number : 2002-215703

(43)Date of publication of application : 02.08.2002

(51)Int.Cl.

G06F 17/50

G06F 9/44

H01L 21/82

(21)Application number : 2001-008043

(71)Applicant : MATSUSHITA ELECTRIC IND CO LTD

(22)Date of filing : 16.01.2001

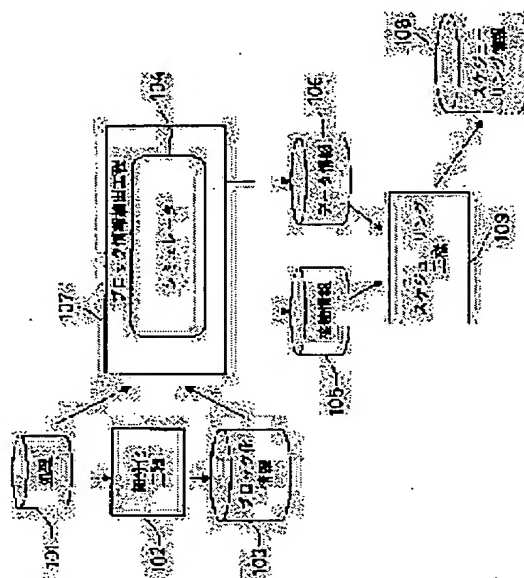
(72)Inventor : YONEDA KEI

(54) HARDWARE/SOFTWARE COOPERATIVE DESIGN METHOD

(57)Abstract:

PROBLEM TO BE SOLVED: To achieve requested performance in the minimum hardware constitution by restricting resources used in a hardware/software cooperative design method from an algorithm descriptive language.

SOLUTION: This method includes a dividing process 102 to divide a process 101 to blocks based on for-sentences and if-sentences to extract block generating information 103, a block information extracting process 107 to conduct simulation by a process simulator 104 based on block generating information and an execution time data base to determine connection information 105 to indicate data dependence relation between the blocks and data information 106 indicating data quantity renewed in processes in each block, and a scheduling process 109 to conduct scheduling for the blocks to the processes in the order of less holding data quantity based on the connection information and the data information for determining scheduling information 108.



LEGAL STATUS

[Date of request for examination]

[Date of sending the examiner's decision of rejection]

[Kind of final disposal of application other than the examiner's decision of rejection or application converted registration]

[Date of final disposal for application]

[Patent number]

[Date of registration]

[Number of appeal against examiner's decision of rejection]

[Date of requesting appeal against examiner's decision of rejection]

[Date of extinction of right]

Copyright (C); 1998,2003 Japan Patent Office

(19) 日本国特許庁 (J P)

(12) 公開特許公報 (A)

(11) 特許出願公開番号
特開2002-215703
(P2002-215703A)

(43) 公開日 平成14年8月2日 (2002.8.2)

(51) Int.Cl.⁷

識別記号

F I

テーマコード(参考)

G 0 6 F 17/50

6 5 4

G 0 6 F 17/50

6 5 4 M 5 B 0 4 6

9/44

9/06

6 2 0 A 5 B 0 7 6

H 0 1 L 21/82

H 0 1 L 21/82

C 5 F 0 6 4

審査請求 未請求 請求項の数4 O.L. (全 12 頁)

(21) 出願番号 特願2001-8043(P2001-8043)

(22) 出願日 平成13年1月16日 (2001.1.16)

(71) 出願人 000005821

松下電器産業株式会社

大阪府門真市大字門真1006番地

(72) 発明者 米田 圭

大阪府門真市大字門真1006番地 松下電器
産業株式会社内

(74) 代理人 100095555

弁理士 池内 寛幸 (外5名)

Fターム(参考) 5B046 AA08 BA02 KA05

5B076 DD03 EC04

5F064 BB09 DD04 HH06 HH08 HH09

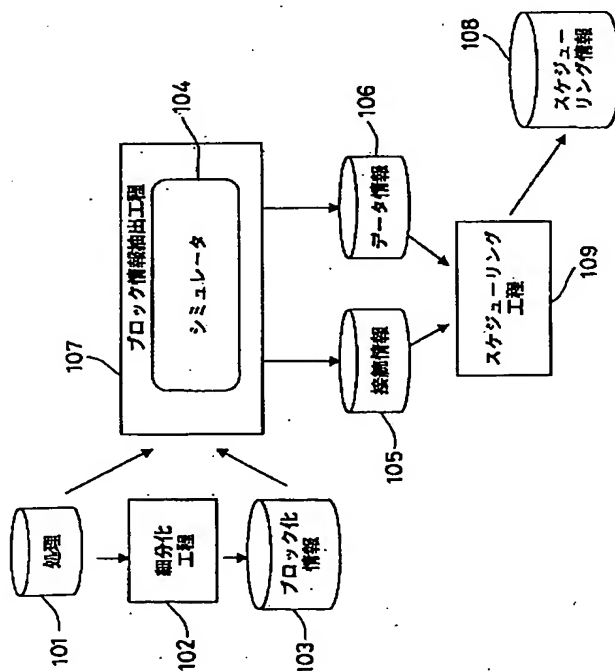
HH10 HH12

(54) 【発明の名称】 ハードウェア/ソフトウェア協調設計方法

(57) 【要約】 (修正有)

【課題】 アルゴリズム記述言語からのハードウェア/ソフトウェア協調設計方法において、使用する資源のリソースを抑え、最小限のハードウェア構成で要求性能を達成する。

【解決手段】 処理101をfor文やif文でブロックごとに細分化しブロック化情報103を抽出する細分化工程102と、ブロック化情報及び実行時間データベースに基づいて処理のシミュレータ104によるシミュレーションを行ない、ブロック間のデータ依存関係を示す接続情報105および各ブロック内の処理において更新されたデータ量を示すデータ情報106を求めるブロック情報抽出工程107と、接続情報およびデータ情報に基づいて、保持するデータ量の少ない処理から順にブロックのスケジューリングを行ってスケジューリング情報108を算出するスケジューリング工程109を含む。



【特許請求の範囲】

【請求項 1】 アルゴリズム記述言語で記述された処理に対するハードウェア／ソフトウェア協調設計方法であって、

前記処理をブロックごとに細分化し、ブロック化情報を抽出する細分化工程と、

前記ブロック化情報に基づいて前記処理のシミュレーションを行ない、前記ブロック間のデータ依存関係を示す接続情報および各ブロック内の処理において更新されたデータ量を示すデータ情報を求めるブロック情報抽出工程と、

前記接続情報および前記データ情報に基づいて、前記ブロックのスケジューリングを行うスケジューリング工程とを含むことを特徴とするハードウェア／ソフトウェア協調設計方法。

【請求項 2】 前記スケジューリング工程において、前記接続情報に基づいて、少なくとも 2 つの処理が並列に実行可能な分岐処理の条件式を検出し、

前記条件式の評価結果に従い実行される全ての処理に対して、前記データ情報を用いて、処理の実行中に保持する必要があるデータ量を算出し、

前記データ量が少ない処理から優先的に実行することを特徴とする請求項 1 記載のハードウェア／ソフトウェア協調設計方法。

【請求項 3】 各々が命令メモリを有しパイプラインステージの処理を担当するマルチプロセッサを用いたパイプライン処理で実行される、アルゴリズム記述言語で記述された処理に対するハードウェア／ソフトウェア協調設計方法であって、

前記処理をブロックごとに細分化し、ブロック化情報を抽出する細分化工程と、

前記ブロック化情報および特定のプロセッサにおける命令コードごとの実行時間をまとめた命令実行時間のデータベースに基づいて、前記処理のシミュレーションを行ない、前記ブロック間のデータ依存関係を示す接続情報、各ブロック内の処理において更新されたデータ量を示すデータ情報、および各ブロック内の処理に要する時間である処理実行時間を求めるブロック情報抽出工程と、

前記接続情報、前記データ情報、前記処理実行時間、および前記パイプライン処理のパイプラインピッチに基づいて、前記ブロックのスケジューリングを行うスケジューリング工程とを含み、

前記パイプラインピッチに過不足なく収まるブロックから優先的に処理を実行し、かつ必要に応じて処理結果を保持するデータ量の少ない処理から優先的に処理を実行することを特徴とするハードウェア／ソフトウェア協調設計方法。

【請求項 4】 各々が命令メモリを有しパイプラインステージの処理を担当するマルチプロセッサを用いたパイ

プライン処理で実行される、アルゴリズム記述言語で記述された処理に対するハードウェア／ソフトウェア協調設計方法であって、

前記処理をブロックごとに細分化し、ブロック化情報を抽出する細分化工程と、

前記ブロック化情報および特定のプロセッサにおける命令コードごとの実行時間をまとめた命令実行時間のデータベースに基づいて、前記処理のシミュレーションを行ない、前記ブロック間のデータ依存関係を示す接続情報、各ブロック内の処理において更新されたデータ量を示すデータ情報、および各ブロック内の処理に要する時間である処理実行時間を求めるブロック情報抽出工程と、

前記処理実行時間および前記パイプライン処理のパイプラインピッチに基づいて、ハードウェアで実現するブロックを抽出し、ハードウェア化情報を算出するハードウェア化工程と、

前記接続情報、前記データ情報、前記処理実行時間、および前記ハードウェア化情報に基づいて、前記ブロックのスケジューリングを行うスケジューリング工程とを含み、

ハードウェア化が必要なブロックについてはハードウェア化を実現しながら、前記パイプラインピッチに過不足なく収まるブロックから優先的に処理を実行し、かつ必要に応じて処理結果を保持するデータ量の少ない処理から優先的に処理を実行することを特徴とするハードウェア／ソフトウェア協調設計方法。

【発明の詳細な説明】

【0001】

【発明の属する技術分野】 本発明は、処理のスケジューリングやハードウェア／ソフトウェアの割り当てに関するハードウェア／ソフトウェア協調設計方法に関する。

【0002】

【従来の技術】 C/C++などのアルゴリズム記述言語で記述されたシステム全体の処理に対し、ASIC等にハードウェア化する部分、CPUで実行させるソフトウェア部分などとシステム全体の処理をハードウェア／ソフトウェアに割り当てる必要がある。いわゆるハードウェア／ソフトウェア協調設計である。ハードウェア／ソフトウェア協調設計では、システム全体の処理を適当なブロックでブロック化しスケジューリングしたものに対し、ハードウェア／ソフトウェアの割り当てを行う。その際に、従来ではハードウェア／ソフトウェア協調設計を行う明確な指針／方法がないため、既存の設計データの流用や設計者の経験や勘に頼るところが大きい。

【0003】

【発明が解決しようとする課題】 まず、従来のハードウェア／ソフトウェア協調設計において既存の設計データを流用した場合、初期の設計コストは減少するが、既存の設計データの仕様制限などを受けるため、システム全

3

体の性能や面積等を考慮した最適なハードウェア/ソフトウェア協調設計を行うことは困難である。そのため、設計段階が進むにつれ新規システムに対する要求性能を満足させることができずに、ハードウェア/ソフトウェア協調設計のやり直しが発生する場合があります、設計コストや設計期間が増大する可能性がある。特に、既存の設計データがハードウェアである場合、そのブロックについては性能、面積、消費電力などが固定値であり、仕様変更などに対して柔軟な対応がとれない。

【0004】また、設計者の経験や勘では、従来までの方法の繰り返しになる場合が多く、システムごとの最適なハードウェア/ソフトウェア協調設計が行われず、性能、面積などの向上が思い通りに図れないことになる。

【0005】つまり、ハードウェア/ソフトウェア協調設計の明確な指針/方法が無い場合、設計プロセスの発展に反して既存データ/設計方法を流用することになり、最適なハードウェア/ソフトウェア協調設計が行われていないというのが実情である。

【0006】本発明は、上記従来の問題点に鑑みてなされたものであり、その目的は、対象とする処理を主にソフトウェアで行なうシステム処理に対して、使用する資源を抑えることが可能なハードウェア/ソフトウェア協調設計方法を提供することにある。

【0007】また、本発明の他の目的は、マルチプロセッサを用いたパイプライン処理によるシステム処理に対して、必要に応じてハードウェアに処理を割り当てながら、使用する資源を抑えかつ要求性能を満たすことが可能なハードウェア/ソフトウェア協調設計方法を提供することにある。

【0008】

【課題を解決するための手段】前記の目的を達成するため、本発明に係る第1のハードウェア/ソフトウェア協調設計方法は、アルゴリズム記述言語で記述された処理に対するハードウェア/ソフトウェア協調設計方法であって、処理をブロックごとに細分化し、ブロック化情報を抽出する細分化工程と、ブロック化情報に基づいて前記処理のシミュレーションを行ない、ブロック間のデータ依存関係を示す接続情報および各ブロック内の処理において更新されたデータ量を示すデータ情報を求めるブロック情報抽出工程と、接続情報およびデータ情報に基づいて、ブロックのスケジューリングを行うスケジューリング工程とを含むことを特徴とする。

【0009】この場合、スケジューリング工程において、接続情報に基づいて、少なくとも2つの処理が並列に実行可能な分岐処理の条件式を検出し、条件式の評価結果に従い実行される全ての処理に対して、データ情報を用いて、処理の実行中に保持する必要があるデータ量を算出し、データ量が少ない処理から優先的に実行することが好ましい。

【0010】この第1のハードウェア/ソフトウェア協

4

調設計方法によれば、細分化しブロック化した処理に対し、各ブロックで更新されたデータ量を評価しながらスケジューリングを行うことで、記憶装置のリソースを抑えることができる。

【0011】前記の目的を達成するため、本発明に係る第2のハードウェア/ソフトウェア協調設計方法は、各々が命令メモリを有しパイプラインステージの処理を担当するマルチプロセッサを用いたパイプライン処理で実行される、アルゴリズム記述言語で記述された処理に対するハードウェア/ソフトウェア協調設計方法であって、処理をブロックごとに細分化し、ブロック化情報を抽出する細分化工程と、ブロック化情報および特定のプロセッサにおける命令コードごとの実行時間をまとめた命令実行時間のデータベースに基づいて、前記処理のシミュレーションを行ない、ブロック間のデータ依存関係を示す接続情報、各ブロック内の処理において更新されたデータ量を示すデータ情報、および各ブロック内の処理に要する時間である処理実行時間を求めるブロック情報抽出工程と、接続情報、データ情報、処理実行時間、およびパイプライン処理のパイプラインピッチに基づいて、ブロックのスケジューリングを行うスケジューリング工程とを含み、パイプラインピッチに過不足なく収まるブロックから優先的に処理を実行し、かつ必要に応じて処理結果を保持するデータ量の少ない処理から優先的に処理を実行することを特徴とする。

【0012】この第2のハードウェア/ソフトウェア協調設計方法によれば、マルチプロセッサのパイプライン処理でシステムのスケジューリングを行う際に、細分化した各ブロックの処理に要する実行時間を算出し、パイプラインピッチに過不足なくブロックをスケジューリングすることで、各パイプラインステージにおけるプロセッサの遊びの時間を少なくすることができる。

【0013】前記の目的を達成するため、本発明に係る第3のハードウェア/ソフトウェア協調設計方法は、各々が命令メモリを有しパイプラインステージの処理を担当するマルチプロセッサを用いたパイプライン処理で実行される、アルゴリズム記述言語で記述された処理に対するハードウェア/ソフトウェア協調設計方法であって、処理をブロックごとに細分化し、ブロック化情報を抽出する細分化工程と、ブロック化情報および特定のプロセッサにおける命令コードごとの実行時間をまとめた命令実行時間のデータベースに基づいて、前記処理のシミュレーションを行ない、ブロック間のデータ依存関係を示す接続情報、各ブロック内の処理において更新されたデータ量を示すデータ情報、および各ブロック内の処理に要する時間である処理実行時間を求めるブロック情報抽出工程と、処理実行時間およびパイプライン処理のパイプラインピッチに基づいて、ハードウェアで実現するブロックを抽出し、ハードウェア化情報を算出するハードウェア化工程と、接続情報、データ情報、処理実行

5

時間、およびハードウェア化情報に基づいて、ブロックのスケジューリングを行うスケジューリング工程とを含み、ハードウェア化が必要なブロックについてはハードウェア化を実現しながら、パイプラインピッチに過不足なく収まるブロックから優先的に処理を実行し、かつ必要に応じて処理結果を保持するデータ量の少ない処理から優先的に処理を実行することを特徴とする。

【0014】この第3のハードウェア/ソフトウェア協調設計方法によれば、ハードウェア化が必要なブロックを抽出し、ハードウェア化を実現しながらスケジューリングを行うことで、対象となるシステムに対して、最小限のハードウェア構成で要求性能を満たすことができる。

【0015】

【発明の実施の形態】以下、本発明の実施の形態について、図面を参照して説明する。

【0016】（第1の実施の形態）図1は、本発明の第1の実施の形態によるアルゴリズム記述言語からのハードウェア/ソフトウェア協調設計方法の構成を模式的に示す図である。図1において、ハードウェア/ソフトウェア協調設計方法は、アルゴリズム記述言語で記述された処理101と、処理を細分化しブロック化する細分化工程102と、細分化工程102より得られる処理のブロック化情報103と、ブロック化情報103をもとにシミュレータ104による処理101のシミュレーションを行い、ブロック間の接続情報105とブロックで更新されるデータ量を示すデータ情報106を抽出するブロック情報抽出工程107と、接続情報105およびデータ情報106に基づいて、ブロックレベルでのスケジューリングを行い、スケジューリング情報108を算出するスケジューリング工程109から構成される。

【0017】図2は、本実施形態においてスケジューリング対象となっているアルゴリズム記述言語（C言語）で記述された処理101の一例を示す図である。

【0018】図3および図4は、図2の処理を細分化工程102より細分化した時のブロック化情報103を表す図である。

【0019】図5は、ブロック情報抽出工程107から得られる図2の処理の接続情報105を表す図である。

【0020】図6は、ブロック情報抽出工程107から得られる図2の処理のデータ情報106を表す図である。

【0021】図7は、図5の接続情報と図6のデータ情報を統合して表した図である。

【0022】図8および図9は、分岐処理の実行順序を変えた場合における各ブロックの処理結果を保持するのに最低限必要な記憶装置の容量を表わした図である。

【0023】図10は、スケジューリング工程109から算出されるスケジューリング情報108を表す図である。

6

【0024】次に、本実施の形態によるハードウェア/ソフトウェア協調設計方法について、図1から図10を用いて具体的に説明する。

【0025】図2で示す処理は、main関数の14行目において、サブモジュールであるfunction関数を呼んでいる算術演算の処理を表す。この処理に対してスケジューリングを行うには、まず処理のブロック化が必要である。細分化工程102では、構文解析を行ってfor文やif文で示される分岐処理の記述とその他の代入文で細分化を行う。つまり図2の処理において、for文、if文、else文などの分岐処理の条件式毎に細分化を行う。

【0026】図2のmain関数およびfunction関数の細分化結果であるブロック化情報103をそれぞれ図3および図4に示す。図3では、7行目のfor文、8行目のif文、11行目のelse文で、細分化を行われ、さらに11行目のelse文の処理が終了する13行目、7行目のfor文の処理が終了する15行目で細分化が行われている。図4についても同様に、function関数について細分化を行っている。なお、図3および図4では、以降の説明のために細分化したブロック毎にA1、A2、…およびB1、B2、…とブロック番号を付加している。

【0027】次に、図2で示す処理101と、図3および図4で示すブロック化情報103とを、ブロック情報抽出工程107に入力し、接続情報105およびデータ情報106を抽出する。以下にその工程を説明する。

【0028】図5は、図4のブロック化情報103を用いてブロック情報抽出工程107より得られた各ブロックごとのデータ依存関係である接続情報105を示す。図5では、function関数の処理を逐次的に処理した場合の実行順に矢印（斜線）を記載している。この図において、ブロックB2の処理結果である“z1

[0]～z1[15]”のデータはブロックB4で利用されるため、処理の接続情報としてブロックB2からブロックB4へ矢印（実線）が記載されている。以下同様に、ブロックB3からブロックB5へはデータ“c”が、ブロックB4からブロックB6へはデータ“d”が、ブロックB5からブロックB6へはデータ“z2[0]～z2[15]”が処理結果として受け渡されている。

【0029】次に、図6は、図4のブロック化情報103を用いてブロック情報抽出工程107より得られる各ブロックで更新されたデータ量であるデータ情報106を示す。例えば、図6において、ブロックB1で更新されるデータ量は“c”、“d”、“e”であり、データ量は3となる。以下同様に更新されるデータ量は、ブロックB2では“z1[0]”～“z1[15]”で16、ブロックB3では“c”で1、ブロックB4では“d”で1、ブロックB5では“z2[0]”～“z2

「15」で16、ブロックB6では「e」で1となる。ここでデータ量の換算方法は整数1個を1としているが、処理によって換算方法は自由に設定できることはいうまでもない。

【0030】図5の接続情報105から、ブロックB2で確定したデータ「z1[0]」～「z1[15]」の値がブロックB4で読み出され、ブロックB3で確定したデータ「c」がブロックB5で読み出されていることがわかる。さらに、ブロックB2とブロックB3の間にはデータの確定/読み出しの関係がないことがわかるため、ブロックB2、B4の処理とブロックB3、B5の処理が互いに並列処理可能であることがわかる。つまり、ブロックB1の後にブロックB2とブロックB3はどちらの処理を先に行っても問題ない。さらに、ブロックB6は、図5よりブロックB4とブロックB5の処理結果を読み出しているため、ブロックB4、ブロックB5の処理が終了しなければブロックB6は処理を開始できない。

【0031】以上のブロック間のデータ依存関係に、図6のデータ情報106を加えたものを図7に示す。図7において、ブロックB2、B4の処理とブロックB3、B5の処理が並列処理可能であり、ブロックB6はこの2つの並列処理の結果を読み出して処理していることがわかる。さらに、図7には、ブロック間のデータ情報も付記して示しており、例えばブロックB2で確定されたデータ「z1[0]」～「z1[15]」のデータ量16がブロックB4で読み出されていることがわかる。

【0032】以下では、図2の関数functionを例に、スケジューリング工程109について説明する。

【0033】関数functionの処理を1つのプロセッサで処理する場合、図7から分かるように、ブロックB1の次にブロックB2、B4あるいはブロックB3、B5のどちらのブロックを優先的に処理するかが問題となる。

【0034】図8は、処理の順番をブロックB1、B2、B4、B3、B5、B6（以下順番Cとする）とした場合における、各ブロックの処理結果を保持するために最低限必要な記憶装置の容量を表したものである。同様に、図9は、処理の順番をブロックB1、B3、B5、B2、B4、B6（以下順番Dとする）とした場合における、各ブロックの処理結果を保持するために最低限必要な記憶装置の容量を表している。図8、図9ともに、当該ブロックが次のブロックへ処理する際に最低限保持しなくてはならないデータのための記憶装置の容量をそれぞれ明記している。

【0035】まず、図8について説明する。図8のブロックB1-B2間においてデータ「c」、「d」、「e」が、またブロックB2-B4間においてデータ「z1[0]」～「z1[15]」が新規に更新され、これらのデータを以降の処理で必要なデータとして保持

する必要がある。しかし、B4-B3間では、データ「d」が新規に更新されるが、ブロックB1-B2間においてデータ「d」を保持する記憶装置は既に用意されているため、データ「d」を保持するための記憶装置は新たに必要でない。ブロックB3-B5間で更新されるデータ「c」についても同様である。そして、ブロックB5-B6間では、データ「z2[0]」～「z2[15]」が新たに更新され、それを保持する記憶装置が必要となる。

10 【0036】次に、図9について説明する。図9では、ブロックB1-B3間において、データ「c」、「d」、「e」が新規に更新され、これらのデータを以降の処理で必要なデータとして保持する必要がある。しかし、次のブロックB3-B5間では、データ「c」が更新されるが、ブロックB1-B3間でデータ「c」を保持する記憶装置は既に用意されているので、データ「c」を保持するための記憶装置は新たに必要ではない。次のブロックB5-B2間では、データ「z2[0]」～「z2[15]」を保持するための記憶装置が新たに必要で、さらにブロックB2-B4間では、データ「z1[0]」～「z1[15]」を保持するための記憶装置が必要となる。しかし、後の処理であるブロックB4-B6間では、データ「z1[0]」～「z1[15]」は必要ないためデータ「z1[0]」～「z1[15]」を保持する記憶装置は必要なくなる。

20 【0037】以上、図8および図9から、処理結果を一時的に保持する記憶装置の容量は、順番Cで処理を行ったほうが順番Dで処理を行うよりも小さいことがわかる。よって記憶装置のリソースの観点から、各ブロックの処理を順番Cで行った方が良いことがわかる。以上より、スケジューリング工程110から得られる関数functionのスケジューリング情報109は、図10に示すものとなり、本実施の形態によれば、記憶装置のリソースが少なくなるように、ブロックのスケジューリングを行なうことが可能になる。

30 【0038】なお、以上のスケジューリング方法は、1つのプロセッサの場合に限らず複数のプロセッサで処理する場合においても同様であることは言うまでも無い。

40 【0039】（第2の実施の形態）図11は、本発明の第2の実施の形態によるアルゴリズム記述言語からのハードウェア/ソフトウェア協調設計方法の構成を模式的に示す図である。図11には、複数の同一のプロセッサが各々命令メモリを備え、このプロセッサが各パイプラインステージの処理を実行するマルチプロセッサのパイプライン処理において、本実施の形態によるハードウェア/ソフトウェア協調設計方法を示している。

50 【0040】図11において、ハードウェア/ソフトウェア協調設計方法は、特定のプロセッサにおける各命令の実行時間をデータベースとしてまとめた命令実行時間のデータベース1101と、パイプライン処理の対象と

なる処理1102と、細分化工程1103より細分化したブロック化情報1104および命令実行時間のデータベース1101に基づいて、処理1102のシミュレータ1105によるシミュレーションを行ない、ブロック間の接続情報1106、ブロック内で更新されるデータ量を示すデータ情報1107、および各ブロックごとの処理に要する時間である処理実行時間1108を抽出するブロック情報抽出工程1109と、接続情報1106、データ情報1107、処理実行時間1108、および設計者の要求するパイプラインピッチ1110に基づいてブロックレベルでのスケジューリングを行い、スケジューリング情報1111を算出するスケジューリング工程1112とから構成される。

【0041】図12は、処理1102における接続情報1106と処理実行時間1108の一例を表した図である。

【0042】図13は、スケジューリング工程1112から得られるスケジューリング情報1111を示す図である。

【0043】図14は、本実施形態においてスケジューリング可能か否かの判断手順を示すフローチャートである。

【0044】次に、本実施の形態によるハードウェア/ソフトウェア協調設計方法について、図11から図14を用いて具体的に説明する。

【0045】図11において、命令実行時間のデータベース1101は、特定のプロセッサの命令、例えば加算、積算などの単純な命令ごとに処理時間をデータベースとしてまとめたものである。これをブロック化情報1104とともにシミュレーションすることで、各ブロックにおける処理実行時間1108を命令実行時間のデータベース1101を参照して算出することができる。ここで、ブロック化情報1104、接続情報1106、データ情報1107は第1の実施の形態で説明した手順と同様な方法で抽出可能である。

【0046】図12は、ブロック情報抽出工程1109から得られる処理実行時間1108と接続情報1106を表す。図12では、処理1102を細分化工程1103より細分化し、1つ以上のブロックで構成される一連の処理ごとにa、b、c、…とし、さらにその一連の処理においてブロック毎にa1、a2、b1、b2、…としている。さらに、各ブロック(a1、a2、b1、b2、…)の長さをそれぞれの処理実行時間1108の大きさに対応させ、並列に実行可能な処理については各ブロックを並列に表記している(例えば、e1、e2とf1、f2は並列に処理可能である)。

【0047】次に、図12で示す処理実行時間1108と接続情報1106をもつ処理1102に対するスケジューリングの方法について説明する。

【0048】まず、パイプラインピッチ1110をスケ

ジューリング工程1112に入力する必要がある。ここで、パイプラインピッチ1110は、パイプライン処理の各ステージで行う処理実行時間を表しており、各ステージの処理実行時間を均一化し、ばらつきを無くすことにより、各ステージの処理に遊びの時間がなくなり、高速なパイプライン処理の設計が可能となる。そこで設計者は、要求する処理性能とパイプライン処理のステージ数からパイプラインピッチ1110を算出し、それをスケジューリング工程1112に入力する。なお、本実施の形態におけるマルチプロセッサのパイプライン処理では、パイプライン処理のステージ数は使用するプロセッサの個数と等価である。

【0049】図12に示すデータ依存関係および処理実行時間を有する処理から、ブロックa1、a2の次に処理可能なブロックの候補として、ブロックb1～b4、c1～c2、およびd1～d4があることがわかる。ここで、ブロックa1、a2の処理実行時間の和がパイプラインピッチ1110よりも小さい場合、つまり同一のパイプラインピッチ1110内にブロックa1、a2の他にさらにブロックの追加が可能である場合、ブロックb1～b4、c1～c2、およびd1～d4の中からいずれかのブロックを選択し追加しなくてはならない。

【0050】本実施の形態によるスケジューリング工程1112では、ブロックa1、a2の次にブロックb1またはブロックc1あるいはブロックd1を挿入し、その時の処理実行時間の総和をそれぞれの場合において算出する。そして、処理実行時間の総和がパイプラインピッチ1110よりも小さい場合には、さらに処理を挿入する。例えば、ブロックa1、a2、b1の処理実行時間の総和がパイプラインピッチ1110よりも小さい場合には、ブロックb1に継続して処理されるブロックb2を挿入し、その時の処理実行時間の総和を算出する。そして算出した処理実行時間の総和がパイプラインピッチ1110よりも大きければ、パイプラインピッチ1110に挿入可能な処理はブロックa1、a2、b1であると確定し、逆に、パイプラインピッチ1110よりも小さければさらにブロックを追加する。

【0051】以上の作業を繰り返すことにより、パイプラインステージにおける処理実行時間の総和がパイプラインピッチ1110よりも大きくならない程度まで可能な限り多くのブロックを挿入する。これを並列処理可能なブロックごとに行い算出した処理実行時間の総和を比較して、最もパイプラインピッチ1110に近い値をもつブロックの順番をスケジューリング情報1111として算出する。

【0052】図13のスケジューリング情報1111は、ブロックa1、a2の次にブロックd1、d2を挿入した時の処理実行時間がパイプラインピッチ1110に最も近い値になったため、ブロックa1、a2、d1、d2をパイプライン処理の第1のステージに行う処

理として確定している。以降、第2のステージでは、第1のステージにおける処理の継続としてブロックd3、d4の処理を実行し、その後の処理については、ブロックc1～c2とブロックb1～b4の中からパイプラインピッチ1110に最も過不足なく収まるものを選択し、スケジューリングを行っている。

【0053】なお、上記のスケジューリング方法において、当該ブロックの次に実行する処理が一意に決定できない場合、例えばパイプラインステージの処理としてa1、a2、d1、d2を行なう場合とa1、a2、c1を行なう場合で処理実行時間の総和が等しい場合には、第1の実施の形態と同様に、処理結果を保持する記憶装置のリソースが少ない処理を、次に処理するブロックとして選択する。

【0054】なお、本実施の形態によるハードウェア/ソフトウェア協調設計方法では、パイプラインピッチ1110よりも処理実行時間が長いブロックがある場合には、本実施の形態のスケジューリングは実行できない。

【0055】図14は、本実施の形態によるスケジューリングが可能か否かの判断を行うフローチャートを示す。各ブロックの処理実行時間Tsとパイプラインピッチi110(Pp)を比較し、その比較の結果、パイプラインピッチ1110よりも処理時間を要するブロックが1つでもあった場合、本実施の形態のスケジューリングは実行できない。これを解決する方法について、第3の実施の形態として次に説明する。

【0056】(第3の実施の形態) 図15は、本発明の第3の実施の形態によるアルゴリズム記述言語からのハードウェア/ソフトウェア協調設計方法の構成を模式的に示す図である。図15には、複数の同一のプロセッサが各々命令メモリを備え、このプロセッサが各パイプラインステージの処理を実行するマルチプロセッサのパイプライン処理において、本実施の形態によるハードウェア/ソフトウェア協調設計方法を示している。図15において、ハードウェア/ソフトウェア協調設計方法は、特定のプロセッサにおける各命令の実行時間をデータベースとしてまとめた命令実行時間のデータベース1501と、パイプライン処理の対象となる処理1502と、細分化工程1503より細分化したブロック化情報1504および命令実行時間のデータベース1501に基づいて、処理1502のシミュレータ1505によるシミュレーションを行ない、ブロック間の接続情報1506、ブロック内で更新されるデータ量を示すデータ情報1507、および各ブロックごとの処理に要する時間である処理実行時間1508を抽出するブロック情報抽出工程1509と、処理実行時間1508および設計者の要求するパイプラインピッチ1510から、ハードウェア化が必要なブロックを抽出し、ハードウェア化情報1511を抽出するハードウェア化工程1512と、接続情報1506、データ情報1507、処理実行時間1

508、ハードウェア化情報1511、およびパイプラインピッチ1510から、ブロックレベルでのスケジューリングを行い、スケジューリング情報1513を算出するスケジューリング工程1514とから構成される。

【0057】図16は、本実施形態のハードウェア化工程1512で行われる処理手順を示すフローチャートである。

【0058】次に、本実施の形態によるハードウェア/ソフトウェア協調設計方法について、図15および図16を用いて具体的に説明する。なお、本実施の形態によるハードウェア/ソフトウェア協調設計方法は、第2の実施の形態に、パイプラインピッチ1510および処理実行時間1508からハードウェア化が必要なブロックを抽出し、ハードウェア化を行うハードウェア化工程1512が追加されたものである。

【0059】図16に示すように、パイプラインピッチ1510(Pp)と処理実行時間1508(Ts)を比較し、パイプラインピッチ1510よりも処理時間が長いブロックについては、パイプラインピッチ1510に収まるように高位合成などによりハードウェア化を行う。図15のハードウェア化情報1511は、この時のハードウェア化されたブロックの情報を示す。このハードウェア化情報1511に基づき、スケジューリング工程1514では、ハードウェア化されたブロックを検出し、1つのパイプラインステージ内にハードウェア化されたブロックを割り当てる。それ以外のハードウェア化が必要ないブロックについては、第2の実施の形態と同様のハードウェア/ソフトウェア協調設計を行う。

【0060】このように、本実施の形態によれば、最小限のハードウェア構成で要求する性能を満たすような処理のハードウェア/ソフトウェア協調設計が行える。ハードウェア構成を極力少なくすることにより、仕様変更などに対してもプロセッサで処理させるプログラムの変更などで柔軟に対応することができる。

【0061】

【発明の効果】以上説明したように、本発明によれば、細分化しブロック化した処理に対し、各ブロックで更新されたデータ量を評価しながらスケジューリングを行うことで、記憶装置のリソースが少ないハードウェア/ソフトウェア協調設計が可能になる。

【0062】また、マルチプロセッサのパイプライン処理でシステムのスケジューリングを行う際に、細分化した各ブロックの処理に要する実行時間を算出し、パイプラインピッチに過不足なくブロックをスケジューリングすることで、各パイプラインステージにおけるプロセッサの遊びの時間が少ないハードウェア/ソフトウェア協調設計が可能になる。

【0063】さらに、ハードウェア化が必要なブロックを抽出し、ハードウェア化を実現しながらスケジューリングを行うことで、対象となるシステムに対して、最小

限のハードウェア構成で要求性能を満たすハードウェア／ソフトウェア協調設計が可能になる。

【図面の簡単な説明】

【図 1】 本発明の第 1 の実施の形態に係るアルゴリズム記述言語からのハードウェア／ソフトウェア協調設計方法の構成図

【図 2】 図 1 のアルゴリズム記述言語で記述された処理 101 の一例を示す図

【図 3】 図 2 の main 関数のブロック化情報 103 を模式的に示す図

【図 4】 図 2 の function 関数のブロック化情報 103 を模式的に示す図

【図 5】 図 1 の接続情報 105 を模式的に示す図

【図 6】 図 1 のデータ情報 106 を模式的に示す図

【図 7】 図 1 の接続情報 105 およびデータ情報 106 を模式的に示す図

【図 8】 図 1 の処理 101 が実行順序 C の場合に必要な記憶装置の容量を示す図

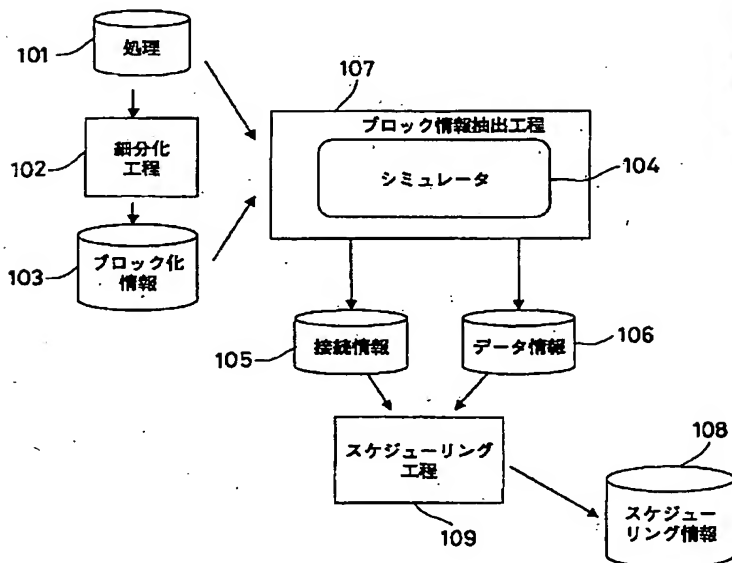
【図 9】 図 1 の処理 101 が実行順序 D の場合に必要な記憶装置の容量を示す図

【図 10】 図 1 のスケジューリング情報 108 を模式的に示す図

【図 11】 本発明の第 2 の実施の形態に係るアルゴリズム記述言語からのハードウェア／ソフトウェア協調設計方法の構成図

【図 12】 図 11 の接続情報 1106 および処理実行

【図 1】



時間 1108 を模式的に示す図

【図 13】 図 11 のスケジューリング情報 1111 を模式的に示す図

【図 14】 本発明の第 2 の実施の形態におけるスケジューリングが可能か否かの判断手順を示すフローチャート

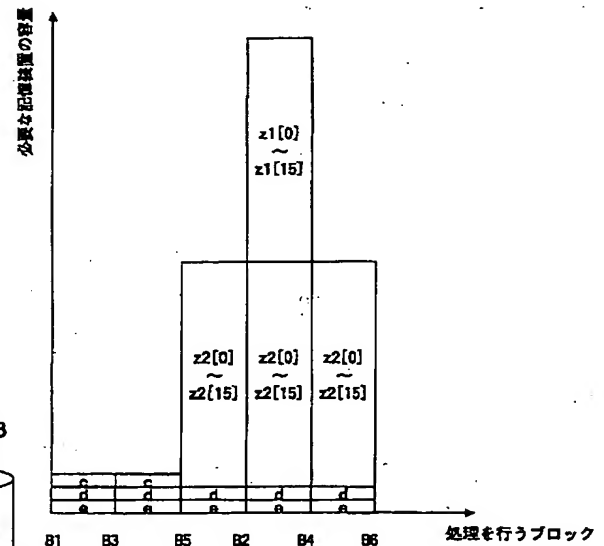
【図 15】 本発明の第 3 の実施の形態に係るアルゴリズム記述言語からのハードウェア／ソフトウェア協調設計方法の構成図

10 【図 16】 図 15 のハードウェア化工程 1514 における処理手順を示すフローチャート

【符号の説明】

101、1102、1502 処理
102、1103、1503 細分化工程
103、1104、1504 ブロック化情報
104、1105、1505 シミュレータ
105、1106、1506 接続情報
106、1107、1507 データ情報
107、1109、1509 ブロック情報抽出工程
20 108、1111、1513 スケジューリング情報
109、1112、1514 スケジューリング工程
1101、1501 命令実行時間データベース
1108、1508 処理実行時間
1110、1510 パイプラインピッチ
1511 ハードウェア化情報
1512 ハードウェア化工程

【図 9】



【図2】

```

1: main()
2: {
3:     int i, a, b, x[16], y[16];
4:     int x={2,7,3,6,9,10,1,5,6,6,13,17,9,4,7,8}
5:     int y={1,4,7,2,5,8,3,6,9,15,16,1,7,7,0,0}
6:     a=0;
7:     for (i=0; i<16; i++) {
8:         if (i<8) {
9:             b=a+2;
10:        }
11:        else {
12:            b=a+1;
13:        }
14:        function(a, b, x, y);
15:    }
16: }

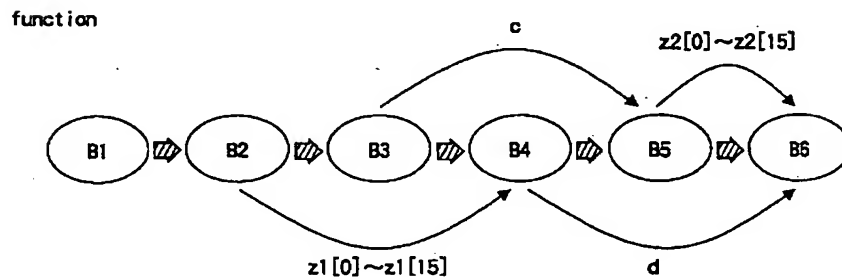
1: function(a, b, x, y)
2: {
3:     int j, k, l, n, c, d, e, z1[16], z2[16];
4:     c=0;
5:     d=0;
6:     e=0;
7:     for (j=0; j<16; j++) {
8:         z1[j]=x[j]*a;
9:     }
10:    for (k=0; k<16; k++) {
11:        c=c+y[k]*b;
12:    }
13:    for (l=0; l<16; l++) {
14:        d=d+z1[l];
15:    }
16:    for (m=0; m<16; m++) {
17:        z2[m]=c*x[m];
18:    }
19:    for (n=0; n<16; n++) {
20:        e=e+z2[n]/d;
21:    }
22: }

```

【図3】

	1: main()
	2: {
	3: int i, a, b, x[16], y[16];
	4: int x={2,7,3,6,9,10,1,5,6,6,13,17,9,4,7,8}
A1	5: int y={1,4,7,2,5,8,3,6,9,15,16,1,7,7,0,0}
	6: a=0;
A2	7: for (i=0; i<16; i++) {
	8: if (i<8) {
A3	9: b=a+2;
	10: }
	11: else {
A4	12: b=a+1;
	13: }
	14: function(a, b, x, y);
B	15: }
	16: }

【図5】



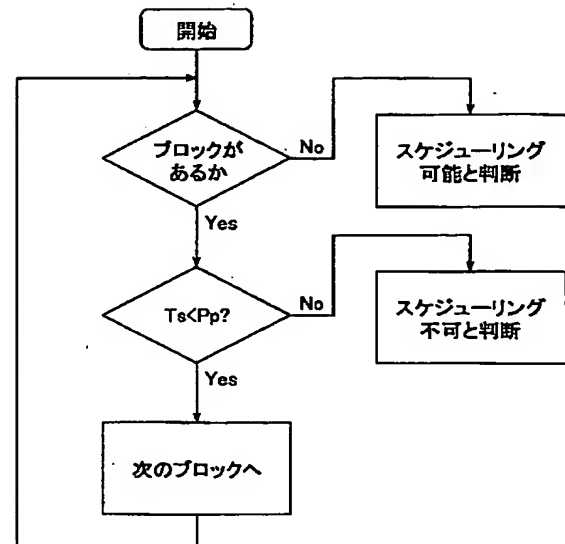
【図 4】

```

1: function(a,b,x,y)
2: {
3:     int j, k, l, m, n, c, d, e, z1[16], z2[16];
4:     a=0;
5:     d=0;
6:     e=0;
7:     for(j=0; j<16; j++) {
8:         z1[j]=x[j]*a;
9:     }
10:    for(k=0; k<16; k++) {
11:        c+=y[k]*b;
12:    }
13:    for(l=0; l<16; l++) {
14:        d+=z1[l];
15:    }
16:    for(m=0; m<16; m++) {
17:        z2[m]=c+x[m];
18:    }
19:    for(n=0; n<16; n++) {
20:        e+=z2[n]/d;
21:    }
22: }

```

【図 14】



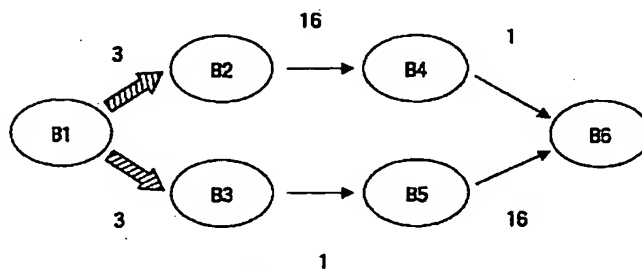
【図 6】

function

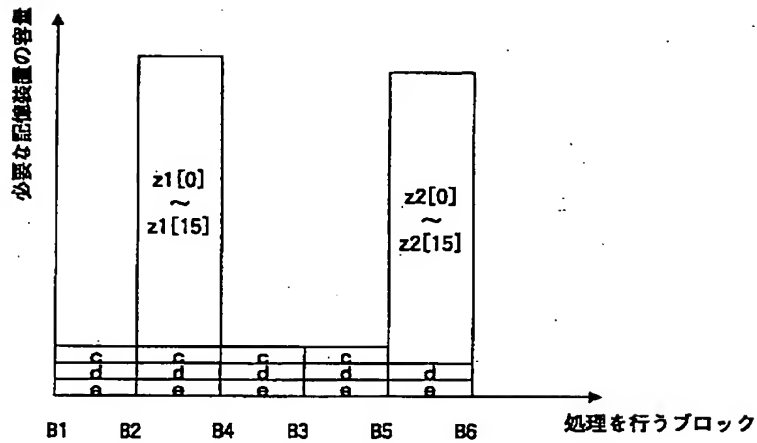


【図 7】

function

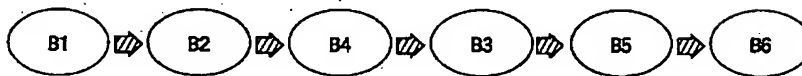


【図 8】

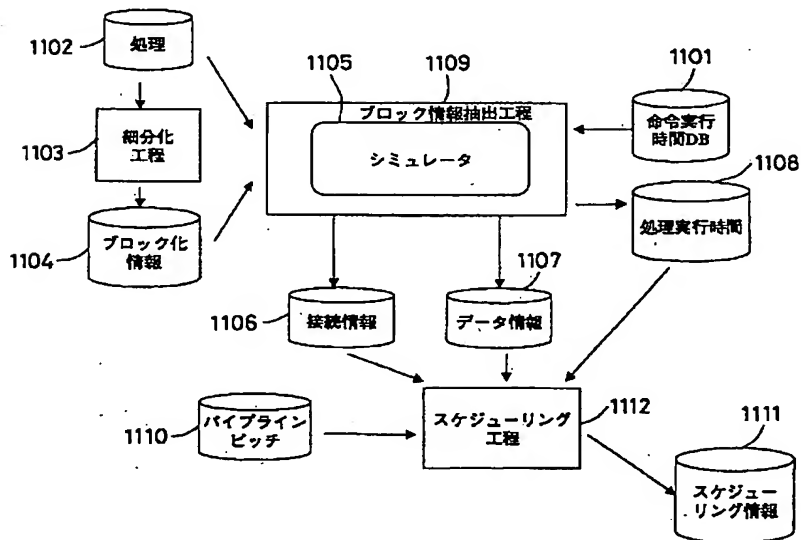


【図 10】

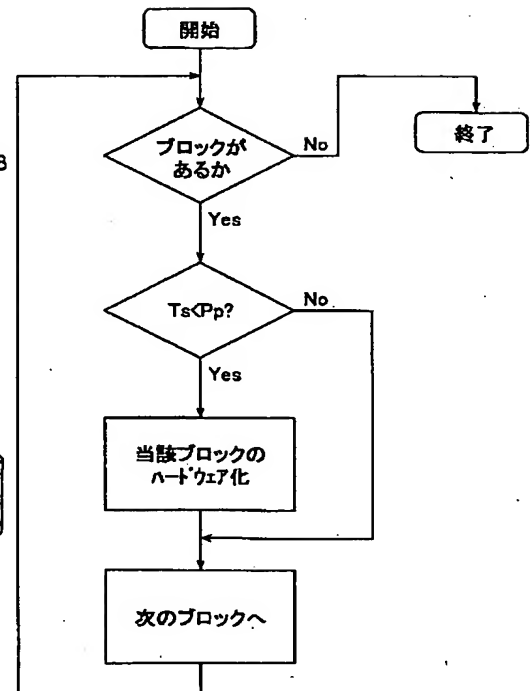
function



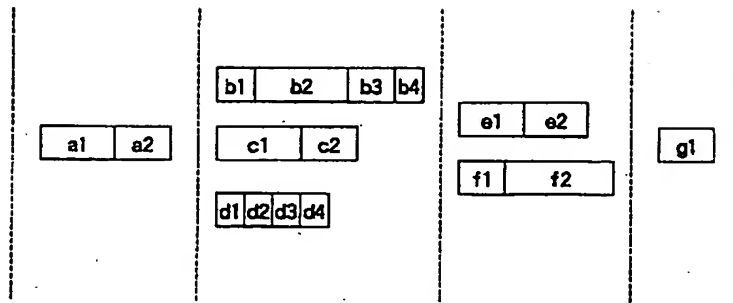
【図 11】



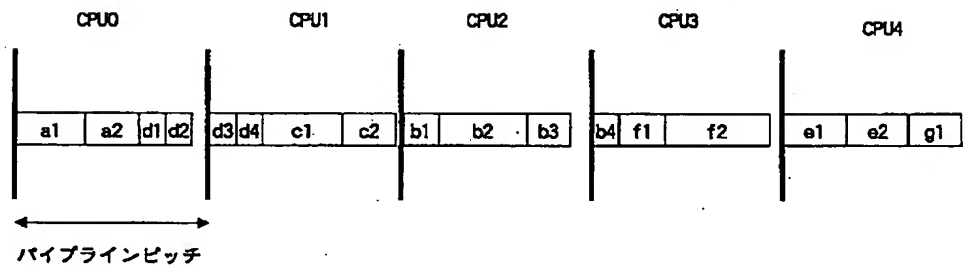
【図 16】



【図 12】



【図 13】



【図 15】

